Docket No.: POU920010138US1

Inventor: Baker et al.

Title: FACILITATING THE USE OF ALIASES DURING THE DEBUGGING OF APPLICATIONS

APPLICATION FOR UNITED STATES

LETTERS PATENT

"Express Mail" Mailing Label No.: ET089520835US
Date of Deposit: March 15, 2002

I hereby certify that this paper is being deposited with the United States Postal Service as "Express Mail Post Office to Addressee" service under 37 CFR 1.10 on the date indicated above and is addressed to: Box Patent Application, Commissioner for Patents, Washington, D.C. 20231.

Name: Sandra L. Kilmer

Signature: Sandra L Kilmer

INTERNATIONAL BUSINESS MACHINES CORPORATION

FACILITATING THE USE OF ALIASES DURING THE DEBUGGING OF APPLICATIONS

Cross-Reference to Related Applications

[0001] This application contains subject matter which is related to the subject matter of the following applications, each of which is assigned to the same assignee as this application and filed on the same day as this application. Each of the below listed applications is hereby incorporated herein by reference in its entirety:

[0002] "Facilitating The Debugging of Applications By Employing User Defined Or Redefined Symbols," Baker et al., Serial No. _____, (IBM Docket No. POU920010136US1), filed herewith; and

[0003] "Generating A Common Symbol Table For Symbols Of Independent Applications," Baker et al., Serial No. _____, (IBM Docket No. POU920010137US1), filed herewith.

Technical Field

[0004] This invention relates, in general, to the debugging of computer applications, and in particular, to facilitating the use of aliases during the debugging of computer applications.

Background of the Invention

[0005] Computer applications (or programs) are created using programming languages. These programming languages include human-readable text, which is translated by a compiler to generate machine-readable instructions. In particular, the compiler translates each line of the programming language into machine-readable instructions, which perform the logical task described by the line of programming language code.

[0006] During the course of programming, errors may occur due to programmer error. These errors include either syntax errors or logic errors. Syntax errors are caught by the compiler, which refuses to generate machine instructions for a line of programming language that it does not understand. Logic errors are caught by detecting that the application does not perform as expected, often resulting in program exceptions or incorrect behavior. While syntax errors can be corrected by determining the correct syntax, logic errors are often much more difficult to ascertain. For this reason, a special class of computer programs has been developed. These programs are called debuggers. One example of a debugger is described in U.S. Patent No. 4,636,940, entitled "Logic Analyzer Using Source Programs Or Other User Defined Symbols In The Trace Specification And The Trace Listing", issued January 13, 1987, which is hereby incorporated herein by reference in its entirety. A debugger allows the user to trace through the user's own application, while it is executing on the computer, to

determine the instance or instances where the logical flow of the application is incorrectly designed.

[0007] A debugger uses debugging information to debug, at runtime, the state of variables of the application. The debugging information describes how the variables of the application are stored and what fields the variables contain. Examples of techniques to generate debugging information are described in U.S. Patent No. 5,715,460 entitled "Template Based Facility For Formatting Compiler Output," issued February 3, 1998; and U.S. Patent No. 5,560,009, entitled "Generating Symbolic Debug Information by Merging Translation and Compiler Debug Information," issued September 24, 1996, each of which is hereby incorporated herein by reference in its entirety.

[0008] In one example, in order to obtain debugging information for a particular variable, the user enters the variable name into the debugger. However, often the variable name is complex and/or long, since it is typically generated by the compiler. Thus, it is useful to allow the user to create a simplified name (i.e., an alias) to reference the variable. This is especially true when the use of aliases is supported by the compiler and/or when the same application is to be debugged multiple times.

[0009] Although previous attempts have been made to allow, in certain limited circumstances, the use of aliases during debugging, a need still exists for an enhanced capability to facilitate the use of aliases during

debugging. In particular, a need exists for a capability that facilitates the use of aliases during debugging, even when the aliases are not known to the debug generation stage of the compiler.

Summary of the Invention

[0010] The shortcomings of the prior art are overcome and additional advantages are provided through the provision of a method of facilitating the use of aliases in debugging of applications. The method includes, for instance, obtaining an alias of an application to be debugged, the alias being unknown to at least a debug generation stage of a compiler of the application; and resolving the alias to a name known to the compiler to facilitate use of the alias in debugging of the application.

[0011] In another aspect of the present invention, a method of facilitating debugging of applications is provided. The method includes, for instance, obtaining an alias of an application to be debugged, the alias lacking associated debug information; and debugging at least a portion of the application using the alias.

[0012] In yet another aspect of the present invention, a method of managing the use of aliases in debugging of applications is provided. The method includes, for instance, determining whether a user attempting to use an alias in debugging at least a portion of the application is authorized to use the alias; and providing debug information

for the alias, in response to the determining indicating authorization.

[0013] System and computer program products corresponding to the above-summarized methods are also described and claimed herein.

[0014] Advantageously, a capability is provided that facilitates the use of aliases during the debugging of applications. In one aspect, the capability enables aliases, which are not known to the debug generation stage of the compiler, to be used during the debugging. This allows users to use simplified names to reference complex symbols without foregoing the debugging of those names.

[0015] Additional features and advantages are realized through the techniques of the present invention. Other embodiments and aspects of the invention are described in detail herein and are considered a part of the claimed invention.

Brief Description of the Drawings

[0016] The subject matter which is regarded as the invention is particularly pointed out and distinctly claimed in the claims at the conclusion of the specification. The foregoing and other objects, features, and advantages of the invention are apparent from the following detailed description taken in conjunction with the accompanying drawings in which:

[0017] FIG. 1a depicts one embodiment of a computing environment incorporating and using one or more aspects of the present invention;

[0018] FIG. 1b depicts one embodiment of a distributed debugger executing within the computing environment of FIG. 1a, in accordance with an aspect of the present invention;

[0019] FIG. 2 depicts one example of an alias table used during debugging of an application, in accordance with an aspect of the present invention; and

[0020] FIG. 3 depicts one embodiment of the logic associated with using the alias table of FIG. 2 to resolve aliases during the debugging of an application, in accordance with an aspect of the present invention.

Best Mode for Carrying Out the Invention

[0021] In one aspect of the present invention, a capability is provided for facilitating the use of aliases during the debugging of computer applications. As one example, the aliases used during the debugging are unknown to the compiler (or assembler) of the application. For example, they are not known to the debug generation stage of the compiler, and thus, debug information is not generated by the compiler for those aliases.

[0022] One embodiment of a computing environment incorporating and using one or more aspects of the present invention is depicted in FIG. 1a. A computing environment 100 includes, for instance, a plurality of computing units 102 coupled to one another via a connection 104. In one example, one of the computing units is a workstation 106 executing an operating system, such as Windows NT or LINUX, as examples, and the other computing unit is a server 108 executing the z/OS or OS/390 operating system offered by International Business Machines Corporation, Armonk, New York. Each computing unit includes, for example, a central processing unit (CPU) 110, memory 112, and one or more input/output devices 114, which are well known in the art.

[0023] Connection 104 is, for instance, a TCP/IP connection, which includes a network interface card 116 at each end. In other embodiments, however, connection 104 may be any type of connection, such as a wire connection, token ring or network connection, to name just a few examples.

[0024] The computing environment described above is only one example. One or more aspects of the present invention can be incorporated and used with other types of computing units, computers, processors, nodes, systems, workstations and/or other environments, without departing from the spirit of the present invention.

[0025] For example, aspects of the present invention can be used in a single system environment. One such environment may include, for instance, an RS/6000 computer

system running the AIX operating system offered by International Business Machines Corporation, Armonk, New York. In yet a further embodiment, one or more aspects of the invention can be included in a large parallel system with a plurality of units coupled to one another. These and many other types of environments are capable of incorporating and using aspects of the present invention.

[0026] In one embodiment, executing within the computing environment is a debugger. As one example, this debugger is a distributed debugger, in which components of the debugger are run on the various computing units. For example, as shown in FIG. 1b, a front-end component 120 of a debugger is executing on workstation 106 and a back-end component 122 of the debugger is executing on server 108. The front-end component includes a user interface to facilitate user input to the debugger; and the back-end component includes the logic used in performing the debugging of a user's program 124 running on server 108. One example of such a distributed debugger is the IBM distributed debugger.

[0027] Although a distributed debugger is described herein, the capabilities of the present invention are not limited to such a debugger. Non-distributed debuggers may also be used. For example, one or more aspects of the present invention can be incorporated and used in a single system environment using a non-distributed debugger.

[0028] A debugger is used during application development to detect one or more errors within the application. The

debugger uses, for instance, debugging information. This debugging information is provided by the compiler, when a debugging option is chosen. As one example, the debugging information includes symbols generated by the compiler that represent each variable or inline definition included in the application source.

[0029] A symbol describes, for instance, the location of a variable in memory at any given time that variable is active in the program. As one example, a symbol includes information about the name of a variable, the type of a variable, and the address or list of addresses for the variable. For example, if the variable int i is declared in a program, a symbol is generated for the variable. The symbol includes, for instance, information about the name (i), the type (int), and information the compiler generates about how to locate the variable.

[0030] In one example, the debugger displays debug information associated with one or more variables of the application. The one or more variables are selected by, for instance, a user debugging the application. Although the selected variables are often defined in the source code version of the application, there is typically a restriction on the variables that can be displayed to the user. For instance, the variables that can be displayed are limited to the strict or formal names for the variables, which have been generated by the compiler, even in those cases where the compiler allows the use of aliases in the application code.

[0031] For example, assume the following aliases are used in the application code:

```
#define A x
#define B y
#define ANOTHER_ALIAS x+y
x=1;
y=2;
B=3;
A=4.
```

In the above example, the programmer uses various aliases, including aliases A, B and ANOTHER_ALIAS. However, the preprocessor of the compiler replaces any occurrences of A, B, and ANOTHER_ALIAS with the variables x, y, and x+y, respectively. Thus, when debugging information is generated by the compiler, the compiler only generates symbols for x, y, and x+y, since the preprocessor has eliminated the values of A, B and ANOTHER_ALIAS. Therefore, when the debugger is used to debug this section of code, the user is not shown debugging information for A, B or ANOTHER_ALIAS, since these aliases are not known to the debug generation stage of the compiler.

[0032] To alleviate the above problem, a capability is provided, in accordance with an aspect of the present invention, that facilitates the use of aliases during debugging of applications, even when the compiler does not generate debug information for the aliases (i.e., the

aliases are not known to the compiler) or even when the aliases are not within the application code.

[0033] In one embodiment, this capability includes adding an alias translation step to the symbolic expression evaluation process of the debugger, in which an alias is resolved to its formal name, such that the formal name can be looked-up in one or more symbol tables to provide debug information for the formal name, and thus, the alias.

[0034] The alias is resolved, in one embodiment, using an alias look-up table 200 (FIG. 2). In one example, alias look-up table 200 includes one or more entries 202, and each entry includes, for instance, an alias name 204 indicating the simplified name chosen by the end user or system administrator; a formal name 206 indicating the name corresponding to the alias that is known to the compiler (e.g., the name that exists in the symbol table created by the compiler); and a user field 208 that identifies one or more users eligible to use the alias name of that entry. User field 208 can include a user name, a user id, an IP address that identifies a remote debug session, as well as other identifying information, as examples. Different users can define their own alias entries (with the same or different alias names) and those entries do not affect other users' definitions. Further, an administrator can define an entry for one or more users.

[0035] In one example, the alias look-up table is initially created from a set of aliases generated from

output of the compiler preprocessor. The alias table is also dynamically maintainable allowing additional entries to be added by the user or system administrator. For instance, entries created by the user at, for instance, runtime may be added. In other embodiments, the alias look-up table can be created and/or maintained in other ways. For instance, it is not necessary to initially create the table from preprocessor output.

[0036] One embodiment of the logic associated with using the alias look-up table to resolve aliases, and therefore, provide debugging information for the aliases, in accordance with an aspect of the present invention, is described with reference to FIG. 3. The logic of FIG. 3 is performed by the debugger, as one example.

[0037] Initially, an expression to be debugged is obtained (e.g., provided, received, have), STEP 300. For example, a user enters the expression into the debugger. Thereafter, the expression is parsed into its various tokens, such as its various operands and operators, STEP 302. The parsing is performed, for instance, by a parser segment of the debugger that parses the expression based on rules associated with the particular programing language used to code the expression.

[0038] Subsequent to parsing the expression or at least a portion of the expression, a determination is made as to whether there are tokens to be processed, INQUIRY 304.

Should there be tokens to be processed, then a token is selected, STEP 306.

[0039] Thereafter, a determination is made as to whether the selected token is in the alias table, INQUIRY 308. If the selected token is not in the alias table, then processing continues with INQUIRY 304 to determine if there are more tokens to be processed. Otherwise, should the token be in the alias table indicating that it is an alias of a variable, then, in one embodiment, a further determination is made as to whether the user debugging the expression is eligible to use the alias, INQUIRY 310. If the user is ineligible to use the alias, then processing once again continues with INQUIRY 304. However, if the user is eligible to use this alias, then the alias is resolved by replacing the token with its formal name, which is obtained from the alias table, STEP 312. Thereafter, processing continues with INQUIRY 304.

[0040] At INQUIRY 304, when there are no more tokens to be processed, then each token is looked up in one or more symbol tables (e.g., created by the compiler, assembler and/or translator) to obtain debugging information, if any, associated with that token, STEP 314. Thus, debugging information is obtained for the tokens of the expression, including any tokens that are aliases.

[0041] In one further embodiment, a formal name for an alias might itself need to be parsed into separate tokens. Should this be necessary or desired, then the process

described above would be repeated for that alias expression. As one example, the recursive parsing is performed prior to the symbol look-up of STEP 314.

[0042] Further, although in the above embodiment user checking is performed for the alias (i.e., INQUIRY 310), in another embodiment, this step may be eliminated.

[0043] Described in detail above is a capability for facilitating the use of aliases during the debugging of applications. In one embodiment, the debugger is enhanced to support the use of aliases by adding supporting resolution techniques to the parser. The added capability is used to determine when an alias has been entered by a user of the debugger and to resolve the alias to the reference variable, which replaces the alias in the expression context. The capabilities of the present invention are not limited to aliases which are created by the compiler, but instead, include aliases added by users at runtime, aliases that are unknown to the compiler, and/or aliases that are a part of other programs, etc.

[0044] Although the embodiments herein refer to a compiler, aspects of the invention are equally applicable to an assembler environment or other like environments. For instance, an alias can be coded in assembler using EQU statements, which provide aliases for absolute symbols. Thus, the term compiler includes degenerate forms of compilers, such as assemblers and other like components.

[0045] Moreover, although the term table is utilized herein, the structure used to hold the alias information is not limited to a table structure. Many types of data structures are usable, and thus, the term table includes those structures.

[0046] The present invention can be included in an article of manufacture (e.g., one or more computer program products) having, for instance, computer usable media. The media has embodied therein, for instance, computer readable program code means for providing and facilitating the capabilities of the present invention. The article of manufacture can be included as a part of a computer system or sold separately.

[0047] Additionally, at least one program storage device readable by a machine, tangibly embodying at least one program of instructions executable by the machine to perform the capabilities of the present invention can be provided.

[0048] The flow diagrams depicted herein are just examples. There may be many variations to these diagrams or the steps (or operations) described therein without departing from the spirit of the invention. For instance, the steps may be performed in a differing order, or steps may be added, deleted or modified. All of these variations are considered a part of the claimed invention.

[0049] Although preferred embodiments have been depicted and described in detail herein, it will be apparent to those

skilled in the relevant art that various modifications, additions, substitutions and the like can be made without departing from the spirit of the invention and these are therefore considered to be within the scope of the invention as defined in the following claims.